CORRECTED VERSION*

CORRECTED VERSION**

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 9/44, 9/46

A1

(11) International Publication Number: WO 99/40512

(43) International Publication Date: 12 August 1999 (12.08.99)

(21) International Application Number: PCT/US99/02801

(22) International Filing Date: 9 February 1999 (09.02.99)

(30) Priority Data:
60/074,143 9 February 1998 (09.02.98) US
09/246,998 8 February 1999 (08.02.99) US

(63) Related by Continuation (CON) or Continuation-in-Part (CIP) to Earlier Application
US 60/074,143 (CON)
Filed on 9 February 1998 (09.02.98)

(71) Applicant (*for all designated States except US*): REUTERS, LTD. [GB/GB]; 85 Fleet Street, London EC4P 4AJ (GB).

(72) Inventors; and

(75) Inventors/Applicants (*for US only*): KLIMCZAK, Jarek [US/US]; 5616 Azure Way, Long Beach, CA 90803 (US). SEWARD, Dan [US/US]; 1720 B East 32nd Street, Austin, TX 78722 (US). ASTIN, Tom [US/US]; 3333 Motor Avenue #303, Los Angeles, CA 90034 (US).

(74) Agent: MAXHAM, Lawrence, A.; Baker & Maxham, Suite 3100, 750 "B" Street, San Diego, CA 92101 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

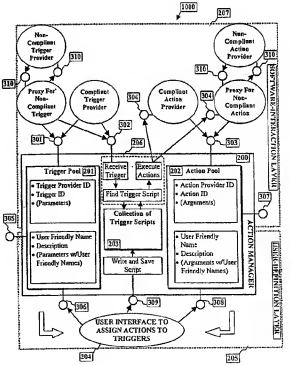
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND SYSTEM FOR USER DEFINED INTERACTIONS BETWEEN PLURALITY OF ACTIVE SOFTWARE APPLICATIONS

(57) Abstract

A method and related computer system having user-interaction that allows assignment of trigger objects to one or more actions objects, both gathered from individual pools (lists) of elements available to the system. "Trigger" is defined as a computer event that initiates execution of a script of actions. The trigger can be: a) a mouse event (clicking on a hypertext link; single and double clicking on an icon, a field, or any program control area; pressing a button); b) a keyboard event (pressing a key, or a sequence of keys); c) a voice generated event and d) any software generated event (conditional; scheduler generated; or any OLE object state). The action can be: a) execution of any software module; b) invocation of a method in a software object; c) setting or querying a property in a software object. Whenever the user activates the trigger, the system invokes the specified actions. The system makes available to a user a way of defining and invoking a series of computer actions once a triggering event occurs; executes the desired actions; maintains and persists (remembers) the table of associated trigger objects and action objects. The invention is preferably implemented with an interactive computer system having a keyboard or other user specified input device, a display, and a user interface capable of initiating execution of one or more application programs. The preferred use of the invention is financial market quote application that incorporates multiple financial service application programs.



*(Referred to in PCT Gazette No. 4/2000, Section II) **(Referred to in PCT Gazette No. 7/2000, Section II)

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD AND SYSTEM FOR USER DEFINED INTERACTIONS BETWEEN PLURALITY OF ACTIVE SOFTWARE APPLICATIONS

TECHNICAL FIELD

The invention relates generally to computer systems, which supports multiple parallel execution of a plurality of application programs ("applications"). In particular, the invention concerns a system and method using a user-definable software interaction system (USIS) through which a user can assign from an action pool and a trigger pool one or more actions (responses) to execute upon occurance of a triggering event (condition).

BACKGROUND ART

The following items are defined as follows: "User" is defined as a person using a computer who has knowledge of the basic functionality of applications providing varying degrees of specialized services (ranging from general, for example a word-processor, to specific, for example a stock market activity monitor). In general, users do not know the art of software programming. A "User-environment" is defined as a configuration of a computer environment (that is hardware, software components, and operating system) that is designed to facilitate the use of the components by a particular user or type of user.

"Software component" is defined as a computer program that enhances the usefulness and functionality in the combined utilization of the computer hardware and operating system; such enhancements include allowing a user to perform tasks not already supported by the operating system and improving upon the performance of tasks already supported by the operating system. A software component can run independently through operating system commands (as do executable files in the Windows operating System, Windows is a trademark of the Microsoft Corporation) or can require invocation by another software component currently being executed (as ActiveX modules do when instantiated in an Object Linking and Embedding (OLE) Container as discussed below). Software components can be used directly by users and other software components. Software components accessed directly by the user through

a user-interface for the purposes of completing specific kinds of tasks are also called applications.

"Interface" is defined as a conceptual and operational communication point (a point where data is passed or shared) between two participants in a system. An interface can be either programmatic or user-driven. A programmatic interface can be implemented in a variety of ways depending upon the computer environment (that is, the operating system, hardware configuration, software configuration). Programmatic interfaces assume that two participants (usually individual software components, but also advanced scripting mechanisms provided by some applications) agree upon the protocol for using the interface, wherein one or both participants expose (that is, make visible and accessible) to the other participant a means of receiving, retrieving, and/or modifying data. A software component can have multiple implementations of the same interface, thus providing a variety means for communication with other participating software components. In this invention, the interfaces used are implemented through conventional methods (that is, Common Object Model (COM) interfaces, C++ methods, and system calls) familiar to those skilled in the art of object-oriented Windows programming (Windows is a trademark of Microsoft Corporation); other conventional methods, however, can be used to implement the interfaces of which the invention consists. A user-driven interface is also referred to as a "user-interface." A software component may or may not have a user-interface, which typically renders or allows manipulation of data relevant to an application. The user-interface includes a variety of means of allowing input from the user and output to the user, such as the graphical representation of data on a computer screen (for example, a document window, a stock market ticker monitor), a graphical representation of means to manipulate application data or other components of the computer environment (for example, buttons, menus, dialog boxes), physical output devices (for example, sound devices, CRT displays), physical input devices (for example, keyboards, mice, voice input mechanisms). In the description below "interface" denotes a programmatic interface, unless specified otherwise.

The invention assumes the existence of a computer environment providing one or more means of allowing the user to input information and receive output through an

interactive user-interface. One example is the graphical user-interface provided by the many modern operating systems. A graphical user interface operates conventionally as a windowing system such as that described in Chapter 13 of V. J. Mayhew's PRINCIPLES AND GUIDELINES IN SOFTWARE USER INTERFACE DESIGN (USA: Prentiss-Hall, 1992), pp. 437-457. Graphical user interfaces, or "GUIs" as they are often designated, have become a popular feature of computers, especially personal computers (PC). One of the many advantages of such GUIs is that they provide a quick and easy platform to display frequently used or required data by selecting and manipulating graphical display elements, such as icons, with a pointing device, such as a mouse. The icons of a GUI are designed to behave in a manner similar to the objects they represent. A GUI/operating system of the preferred embodiment reside within a computer-readable media that allows one or more users to initiate the manipulation of displayed object icons and text on a device. Any suitable computer-readable media may retain the GUI and operating system. The Apple MacIntosh operating system, Microsoft Windows 98 or NT operating system, and UNIX X-Windows are common and very popular examples of GUIs, illustrating the fact that the advantages of GUIs over conventional text-based user interfaces are widely recognized. As a result of most PC using GUIs, a variety of different means for organizing and navigating through various applications have been developed. The operational capability of a user interface of the windowing system type is fundamental to, but distinct from, the invention. The GUI includes conventional ways for initiating and managing application program execution and for initiating, managing, activating, and deactivating application windows on the display. Apple and Macintosh are trademarks of Apple Corporation.

The invention assumes the existence of a computer environment providing one or more means of communication among software components through conventional programmatic interfaces that can be tailored to specific purposes. An exemplary embodiment enhances and customizes a conventional means of inter-software communication present in the Windows operating system environment: "Object Linking and Embedding" (OLE) automation by Microsoft Corporation of Redmond, Washington, is defined as a system of protocols and interfaces giving software component developers the ability to create applications that expose objects to

programming tools and macro languages for creating and manipulating objects exposed in one application from another application and to create tools that access and manipulate objects. OLE automation represents custom controls, which are part of the Windows libraries suite. Custom controls provides programmers with the facility to define commands, write program modules that correspond to the commands, and to list and describe each Windows "instance" (that is, each manifestation of a template of a Windows entity, such as a GUI window). Custom control supports the concept of classes of instances, wherein each instance can be assigned to, and hence be described in part by, one of a plurality of different instance type classes. A class is a template for creating actual functioning of objects of a given type. The objects that have been created of a given class are called instances of that class. Illustrative examples of well known Windows custom controls are Visual Basic eXtensions (VBX) and Object Linking and Embedding (OLE) Control eXtensions (OCX). Another example of a GUI system with an analogous approach to OLE automation in compound document architectures is the OpenDoc controls by Apple Computer, Inc. of Cupertino, CA (OpenDoc is a trademark of Apple Corporation).

Many commercially available GUIs for various applications are based upon windowing schemes, which support simultaneous execution of several applications. Each executing application may be provided with one or more windows which affords users a way to provide input to the executing application and to receive output from it. In the available windowing schemes, even though several applications can be running simultaneously, each updating their respective windows, the complex coordination of multiple applications of different types of architectures within the user-environment is limited to programmatic manipulation of interfaces provided by software components. One particular user-interface, the Microsoft Windows (trademarks of Microsoft Corporation) operating system, has provided a means of software interaction through OLE automation. One problem OLE-type automation poses is the difficulty for users to define complex reactions from a triggering event. Since the user generally does not have the capability of writing batch or script programming for an application, the user cannot script sophisticated interaction among applications in the user-environment. The same is true of macro languages (such as, WordBasic, a scripting language provided in

Microsoft WORD application program) that allow advanced users to program interfaces provided by software components. Typically, users are not able to script complex interaction among different types of software components without using programmatic means.

In the available windowing schemes, a method for allowing user to customize simple procedures is often provided. These tools are, however, too simple to script sophisticated interaction among a plurality of software components. One particular user-interface, the Microsoft Windows (trademarks of Microsoft Corporation) operating system, has provided in its applications the ability to associate an action (response) with a triggering input event such as a single "click" on an icon, pressing a button, pressing a hot or accelerator key (The customization dialog-boxes provided in Microsoft applications, such as WORD and EXCEL, are examples of this form of user-definable software customization). One problem with this form of user-definition is that scripting is often limited to a one-to-one relationship between triggering events and subsequent actions. Another problem is that these user-definition mechanisms do not provide a means of incorporating triggers and actions provided by software components yet to be included in the user-environment. Sometimes users can "record" macros to allow scripting of multiple actions. One problem with this form of scripting is that, since users can only record those actions immediately available in the user-environment, they are limited to scripting those types of actions that are provided at the time the macro is created. In other words, a macro must be completely recreated, if the user wants to include actions and triggers provided by software components added to the user-environment after the original recording of the macro. Moreover, this kind of scripting does not allow more sophisticated interaction among software components, such as passing data not known at the time the macro is recorded or selectively choosing software components based upon the context in which a triggering event occurs. In other words, macro recording takes into consideration the contextual elements existing at the time of script creation, but not the contextual elements existing at the time of script execution.

The present invention provides a solution to these problems by allowing users to non-programmatically script sophisticated software interaction. This concerns

interaction among software components that includes the triggering of not only a singular action, but also series of actions that may or may not depend upon the contextual information available in the user-environment (such as, property values of software components or the state of activity in software components) at the time of execution.

DISCLOSURE OF INVENTION

The primary advantage of the present invention is to provide an efficient and effective action manager method that allows the user to assign a list of one or multiple actions (responses) to a single triggering event (condition), wherein the actions and triggering events may occur in multiple software components not directly aware of each others existence.

The invention comprises a programming method and related system of user-interaction that allows assignment of a trigger elements to one or more actions elements, both gathered from individual pools (lists) of elements available to the system. "Trigger" is defined as a computer event that initiates execution of a script of actions; a Trigger can be: a) a mouse event (clicking on a hypertext link; single and double clicking on an icon, a field, or any program control area; pressing a button); b) a keyboard event (pressing a key, or a sequence of keys); c) a voice generated event and d) any software generated event (conditional; scheduler generated; or any OLE object state). "Action" is defined as a process or function that can be performed by a software component. An action can be: a) execution of any software module; b) invocation of a method in a software object; c) setting or querying a property in a software object. Whenever the user activates the trigger, the system invokes the specified actions. The system makes available to a user a way of defining and invoking a series of computer actions once a triggering event occurs; executes the desired actions; maintains and persists (remembers) the table of associated trigger objects and action objects (an association also known as a "Trigger Script"). The trigger and action objects are also referred to herein as triggers and actions. Moreover, the invention provides an executive means for generating trigger providing software components and action providing software components for assignment and execution at a users disposal by implementing

programming interfaces making available trigger and actions objects. The invention is preferably implemented with an interactive computer system having a keyboard or other user specified input device, a display, and a user interface capable of initiating execution of one or more application programs. The preferred use of the invention is financial market quote application that incorporates multiple financial service application programs.

BRIEF DESCRIPTION OF DRAWING

The foregoing and other objects, aspects, and advantages of the invention will be better understood from the following detailed description of the preferred embodiment of the invention with reference to the drawing, in which:

FIG. 1 is a diagram illustrating a user-definable interaction system using the invention;

FIG. 2 is a diagram illustrating the method of assigning and configuring trigger scripts for selected action objects used in the system of FIG. 1;

FIG. 3 is a diagram illustrating the method of executing the trigger scripts that are assigned as configured by the method in FIG. 2;

FIG.4 is a block diagram illustrating an exemplary system for using the invention;

FIG.5 is a block diagram illustrating a terminal as used in FIG.4;

FIG.6 shows an exemplary container application for using the invention in financial market quote application used in the exemplary system of FIG. 4;

FIG.7 shows a way for inputting a trigger object to the action manager system in the financial application shown in FIG. 6;

FIG.8 shows a continuation of the process shown in FIG. 7; and

FIG.9 shows a completion of the process shown in FIG. 8.

BEST MODE FOR CARRYING OUT THE INVENTION

I. System Description: Referring to FIG. 1, a user-definable interaction software system (USIS) 1000 is driven by two basic elements: *trigger objects* and *action objects*. These elements each operate on two layers: the software-interaction layer 207 and the user-definition layer 205. The USIS uses Component Object Model (COM) technology that is well known in the software arts to implement the present invention using the functionality and capabilities of various operating system platforms (for example, Microsoft Windows and Apple operating system (Microsoft and Windows are trademarks of Microsoft Corporation in Redmond, Washington and Apple is a trademark of Apple, Inc., Cupertino, CA). Those of skill in the relevant arts will readily understand from the following description of how to implement the present invention using the functionality and capabilities of Microsoft Windows based operating system in the preferred embodiment and how to adapt the present invention to other operating system environments, such as Apple or UNIX based machines. As such, the following description of the invention will include only such detail as is necessary to understand the implementation of the present invention, but will not include a detailed description of the elements of Microsoft Windows that are as an exemplary and preferable operating system platform used to implement the present invention in such detailed descriptions of these elements that are readily and publicly available. The basic concepts, which are necessary for understanding the system components and methods, are as follows.

Trigger Objects: Trigger objects are unique events occurring in participating module objects. Such events can take a variety of forms that include but are not limited to: a) the results of user defined activity that include pressing a function key, clicking an on-screen button, selecting an item from a menu, entering information into a field, issuing a voice command, or other user defined event; b) notifications from non-user input sinks that include information from network data streams (for example. when a keyword is found in a data packet from a news feed), operating system messages (for example, when a window is activated), common software interaction protocols (for example, when a COM method is called or a specific type of DDE message is received), proprietary messaging protocols (for example, when a scheduling program broadcasts an alarm); and c) changes in monitored elements of the global environment that include

network connections (for example, whether a certain server remains available), the system clock (for example, whether a specific amount of time has elapsed), and a state of other applications (for example, whether an Internet browser has been closed).

Action Objects: Action Objects are unique commands for executing specific software functionality. Such activity can include a variety of forms that include but are not limited to: a) execution of commands inherent in the global environment--such as, issuing standard operating system commands (for example, executing modules, opening documents, closing windows), following commonly supported software-interaction protocols (for example, creating objects through COM), setting accessible system variables (for example, printer defaults); and b) execution of commands provided by specific applications or modules that include calling exported functions (for example, OLE automation methods, dynamic link libraries (DLL)), setting public variables (for example, OLE Object Properties, currently open document, currently open user resource locator (URL)).

Layers of Interaction: Triggers and actions operate at two layers of abstraction, both of which are necessary for allowing user-definable software scripting that include: a) Software-Interaction Layer that at this level, triggers and actions are uniquely identified elements of participating software modules as described below. (Note that interaction at this layer denotes interaction among software components, not interaction between the user and a software component.); and b) User-Definition Layer that at this level, triggers and actions have meaningful identifiers (that is, user-friendly names) that allow end-users to script activity in their user-environment through a non-programmatic interface.

These layers are coordinated by the Action Manager 200 components as discussed below. The Action Manager provides interfaces for allowing users to create trigger-action object relationships at the User-Definition Layer (that is, without users having to know the underlying software configuration or architecture). The Action Manager also provides software interfaces for communicating with participating components. Finally, the Action Manager translates user-meaningful identifiers for trigger and action objects to and from software-meaningful identifiers for triggers and actions.

II. System Components: USIS 1000 is composed of three types of components: Trigger Providers, Action Providers, and Action Manager. (Note that the Action Manager must be a single instance of a single module, but that Providers (especially, but not exclusively, in object models, such as COM) can have multiple instances of a single class of Provider. Unless otherwise noted, references to "Providers" denote a class of Provider and not a specific instance. Action Manager 200 interacts with multiple Trigger Providers through one or more triggering protocols. Action Manager interacts with multiple Action Providers through one or more activation protocols. Trigger Providers and Action Providers supporting one or more of the protocols supported by Action Manager are considered *compliant*. A protocol determining compliance makes the following assumptions: a) Providers can be distinguished by either class or instance according to a public identifier that include but are not limited to class ID in OLE, class name in OLE, file names (for example, for executable programs), references to specific instantiations (for example, a process identifier, to specific windows of running applications, to specific objects in object containers); b)each Provider's elements (that is, triggers or actions) are distinguished by a communicable identifier that include but are not limited to a system ID in COM, an integer value, a unique string; c) Action Manager 200 has access to both Provider identifiers and element (that is triggers and actions) identifiers through one or more interfaces shared with Providers that include but are not limited to the system registry (for example, the OLE class database), common interfaces for software interaction (for example, COM interfaces), proprietary interfaces for software interaction (for example, a specific OLE automation method, a call to a function in a shared library, a specific window message); d) Trigger Providers can notify the Action Manager 200 when specific events occur through a shared interface that include but are not limited to common interfaces for software interaction (for example, COM interfaces), proprietary interfaces for software interaction (for example, a specific OLE event, a specific window message); and e) Action Manager 200 has means of invoking actions in Action Providers through a shared interface that include but are not limited to the operating system (for example, executing an module, opening a document), common interfaces for software interaction (for example, COM interfaces), proprietary interfaces for software

interaction (for example, a specific OLE automation method, a call to a function in a sharable library, a specific message).

Trigger Provider Proxies and Action Provider Proxies enable non-compliant Trigger Providers and Action Providers to participate in the system. Proxies support both Action Manager compliant protocols and other protocols for interacting with non-compliant Providers. The protocols supported by both Providers and Action Manager operate at the Software-Interaction Layer. The Action Manager also supports protocols that operate at the User-definition Layer. How specific system components implement USIS protocols and interfaces is described in more detail below.

Action Manager: The Action Manager 200 establishes the connection between the Software-interaction Layer and the User-definition Layer by associating Provider and element identifiers with identifiers meaningful in a specific user-environment. The user-environment identifiers enable scripting of specific types of software interaction without requiring the user to know about the underlying software architecture and configuration. In order to manage this software interaction the Action Manager must support protocols for software interaction, for user definition, and for translation between the Software-interaction Layer and the User-definition Layer. The Action Manager supports the following protocols at the layer of software-interaction: a) a means of identifying Trigger Providers; b) one or more interfaces and protocols for gathering trigger identifiers from available Trigger Providers; c) one or more interfaces and protocols for receiving notification of triggered events from Trigger Providers; d) a means of identifying available Action Providers; d) one or more interfaces and protocols for gathering action identifiers from Action Providers; and e) one or more interfaces and protocols for using the action identifiers to invoke specific responses in Action Providers.

At the layer of user-definition, the Action Manager supports one or more interfaces and protocols for creating, editing, and deleting Trigger Scripts as discussed below. Finally, the Action Manager includes three sub-components for translating identifiers of triggers and actions to and from different layers of interaction: a) a Trigger Pool, b) an Action Pool, and c) a collection of Trigger Scripts.

The Trigger Pool: A Trigger Pool 201 enables the translation of triggers between the User-definition Layer and the Software-interaction Layer by associating Trigger Providers and identifiers for their triggers with identifiers tailored to the user-environment. The Trigger Pool is essentially a list of trigger objects available to a specific user-environment, wherein each list item includes: a) identifiers requisite for software-interaction protocols that include i) an identifier for a Trigger Provider (class or instance), ii) an identifier for a specific trigger within the Trigger Provider identified, and c) an optional list of parameters that can be obtained from the trigger; and b) identifiers requisite for user-definition protocols of a user-meaningful identifier for the trigger and (optionally) an identifier for the Trigger Provider.

The Action Manager supports the following functionality for maintaining the Trigger Pool: a) means of learning about software-defined identifiers for triggers and Trigger Providers available in the global environment (that is, providing one or more interfaces for learning about available triggers; b) one or more selective interfaces 305 that are programmatic (for example, a COM interface, a DDL) and/or user-driven (for example, a dialog box, a menu command.) for generating the Trigger Pool 201 (that is, assigning user-meaningful identifiers to software-defined identifiers and storing to a central list); c) one or more means of saving the current state of the Trigger Pool and restoring that state at a later time and another location (for example, reading and writing a file, serializing to and from an OLE compound document, reading and writing system registry entries); and d) one or more interfaces 306 used to query the Trigger Pool for its content, in particular the user friendly identifiers for available triggers.

The Action Pool: The Action Pool 202 enables the translation of actions between the User-definition Layer and the Software-interaction Layer by associating Action Providers and identifiers for their actions with identifiers tailored to the user-environment. The Action Pool is essentially a list of actions available to a specific user-environment, wherein each list item includes: a) identifiers requisite for software-interaction protocols that include an identifier for an Action Provider, an identifier for a specific Action within the Action Provider identified, and an optional set of arguments that can be passed to the Action Provider; and b) identifiers requisite for user-definition protocols that include a user-meaningful identifier for the Action and (optionally) an

identifier for the Action Provider, and a directive on which instance of an Action Provider class should perform an action--that is, a new instance, the first available, or a specific instance.

The Action Manager supports the following functionality for maintaining the Action Pool that include: a) means of learning about software-defined identifiers for actions and Action Providers available in the global environment (that is, providing one or more interfaces for learning about actions); b) one or more programmatic and/or user-driven interfaces 307 for populating the Action Pool (that is, assigning user-meaningful identifiers to software-defined identifiers and storing to a central list); c) one or more means of saving the current state of the Action Pool and restoring that state at a later time and another location (for example, reading and writing a file, serializing to and from an OLE compound document, reading and writing system registry entries); and d) one or more interfaces 308 used to query the Action Pool for its content, in particular the user friendly identifiers for available actions.

Trigger Scripts: The Trigger Scripts associate specific trigger identifiers with one or more specific action identifiers. The Action Manager 200 maintains a collection of Trigger Scripts, allowing them to be saved in some form (for example, in a compound document, in the system registry). Each trigger script includes information necessary for software interaction, including: a) an identifier for an instance of a Trigger Provider; b) an identifier for a specific trigger offered by the identified Trigger Provider; and c) a list of actions, wherein each list item includes an identifier for an Action Provider (including information specifying the scope of activation [for example, within a new instance of an Action Provider, within a specific previously created instance of an Action Provider]), an identifier for a specific action in the Action Provider, and an optional set of parameters.

The Action Manager 200 supports the following functionality for maintaining Trigger Scripts that include: a) one or more interfaces 309 (that is, selectively programmatic and user-driven) for creating Trigger Scripts (that is, allowing users to associate a user-meaningful identifier for a trigger with one or more user-meaningful identifiers for actions and then storing the corresponding software-meaningful identifiers in a list, wherein the translation between user-meaningful and software-

meaningful identifiers refers to the Trigger Pool and the Action Pool created for a specific user-environment); and b) one or more means of saving and restoring the Trigger Scripts at a later time and another location (for example, reading and writing a file, serializing to and from an OLE compound document, reading and writing system registry entries).

Trigger Providers: The Trigger Provider is a software component that maintains a list of events each associated with a unique identifier. Compliant Trigger Providers support one or more of the trigger protocols also supported by the Action Manager. Non-compliant Trigger Providers can participate in USIS 1000 through a Trigger Provider Proxy, which translates protocols not supported by the Action Manager to and from those which are supported. Instantiations of compliant Trigger Providers and Trigger Provider Proxies must each include the following: a) a list of scriptable triggers (for example, mouse-click of an on-screen button, selection from a menu, receipt of an updated message) each associated by an identifier compatible with one of the protocols supported by Action Manager (for example, OLE ID, integer value, unique string); b) one or more compliant (that is, supported by a protocol also supported by Action Manager) interfaces for publishing the list of identifiers for scriptable triggers to the Action Manager; and c) one or more compliant interfaces for notifying the Action Manager of the identifier of a scriptable trigger when the event associated with it has occurred.

Action Providers: The Action Provider is a software component that maintains a list of commands each associated with a unique identifier. Compliant Action Providers support one or more of the activation protocols also supported by the Action Manager. Non-compliant Action Providers can participate in USIS 1000 through an Action Provider Proxy, which translates protocols not supported by the Action Manager to and from those which are supported. Instantiations of compliant Action Providers and Action Provider Proxies must each include the following: a) a list of scriptable actions (for example, opening a document, drawing a chart, making a request for data available over a network, and navigating to a specific URL) each associated by an identifier compatible with one of the protocols supported by Action Manager (for example, OLE ID, integer value, and unique string); b) one or more compliant interfaces for publishing

the list of identifiers for available actions to the Action Manager; and c) one or more compliant interfaces for allowing the Action Manager to use an action identifier to invoke a specific command.

III. Action Manager In System Configuration: In a preferred embodiment of the invention, the Action Manager 200 is implemented as a C++ object using C++ programming language within an OLE container application designed for execution within the Windows operating system using well known OLE programming techniques. A "container" is defined as a software component having multiple contained applications. As is well known in the art, this container application is associated with menu bars, optional scrolling view bars and partitioned display areas. Within these latter display areas, there can be text, graphics, multiple applications, and the like. An example of this is Microsoft WORD word processing program (a trademark of Microsoft Corporation). The containing application is both a Trigger Provider and an Action Provider, for which the Action Manager object provides a proprietary interface for executing USIS 1000 protocols within a software module. The Action Manager 200 is also an OLE automation object participant, using OLE automation methods and events for executing the USIS protocols using multiple independent software modules.

The Action Manager sub-components are also implemented as C++ objects. There is not a one-to-one correspondence between sub-components (that is, the Trigger Pool, the Action Pool, and Trigger Scripts); rather, this preferred embodiment of the Action Manager stores a collection of C++ objects corresponding to Providers. (Note that a single Provider object can be both a Trigger Provider and an Action Provider, hence the use of a single object to implement both types of interfaces). Each of these Provider objects is uniquely identified to the Action Manager by the concatenation of a) a string representing the OLE Program ID, which corresponds to the identifier for a class of Provider, (for example, "Navigator"; "DynamicQuote") and b) a string representing the object name, which corresponds to a specific instance of a class of Provider (for example, "Services" as an instance of the Navigator class; "Commands" as another instance of the Navigator class; "DynamicQuote.2" as the second instance of the Quote class).

Aside from providing a unique key for locating the Provider object within the mapped collection (which is an ordered and searchable object containing other uniquely identified objects--see Microsoft Foundation Classes (MFC) documentation), this concatenation serves as the software-meaningful identifier for a specific Provider. This mapped collection implements part of the functionality of both the Trigger Pool and Action Pool, each of which must store software-meaningful identifiers for Providers. User-meaningful identifiers for the providers can be retrieved by using the Program ID to lookup a user-friendly name in the system registry using the Windows operating system; alternatively, the object name can be used as a user-meaningful identifier.

Each Provider object stores a collection of user-meaningful identifiers for the actions its supports (if any exist). A software-meaningful identifier is mapped to each user-meaningful identifier. This mapped collection implements the remaining functionality of the Action Pool, which, stores action identifiers both for the Software-interaction Layer and for the User-definition Layer.

Each Provider object also stores a mapped collection of software-meaningful identifiers for the triggers it supports (if any exist). A "DWORD" value serves as a unique key for locating the trigger within the Provider objects trigger collection, as well as a software-meaningful identifier representing part of the functionality of the Trigger Pool. Although the Provider object does not store the user-meaningful identifier (another element of the Trigger Pool) within this collection, information in the trigger collection and its parent object (that is, the Provider object) is used to retrieve a user-meaningful identifier for a trigger at the time it is configured (that is, the software-meaningful identifiers for the Trigger Provider and the trigger are used to request the user-meaningful identifier of a trigger from the Trigger Provider itself).

In addition to performing the functionality of the Trigger Pool, the trigger collection also implements this embodiment's version of Trigger Scripts. To each "DWORD" value in the trigger collection, the Action Manager 200 maps a collection of scripts, each of which specifies a software-meaningful identifiers for both an Action Provider and a specific action. Each script is optionally associated with a parameter collection (that is, an ordered collection of data elements, each of which can be passed as an argument to the Action Provider for the specific action).

The mapped collections that implement the functionality of the Trigger Pool, the Action Pool, and Trigger Scripts are persisted in a compound document managed by the application containing the Action Manager object. This document stores changes made by the user through user-interfaces, as well as those made by software components through programmatic interfaces. Note that the Pools and the Scripts have some default configurations (that is, values that are inserted into the document at the time of object creation, but that are editable after initiation) that are read from a proprietary location of the Windows operating system registry or other programmatic interfaces for modifying the Action and Trigger Pools and creating Trigger Scripts as discussed below.

Trigger Providers: In the preferred embodiment of the invention, compliant Trigger Providers take the form of a) OLE automation controls or b) the application containing the Action Manager object. Compliant Trigger Providers maintain a collection of software-meaningful trigger identifiers (a DWORD value in this embodiment), each of which is associated with a specific event. Trigger Providers make themselves visible to the Action Manager via the trigger publication interface 301, trough which they pass a DWORD value identifying a trigger to the Action Manager object 200, which stores the DWORD value in the Trigger Pool 201 as described above. This interface 301 has two implementations in this embodiment of USIS: a) a set of C++ methods exposed by the Action Manager within a software module (that is, within the application containing the Action Manager); and b) a set of proprietary OLE Automation events exposed by compliant Trigger Providers to the Action Manager.

Trigger Providers notify the Action Manager when an event which is associated with a trigger has occurred via trigger notification interface 302, through which they pass a DWORD value identifying the event triggered to the Action Manager object 200, which uses the DWORD value and the identifier of the Trigger Provider to locate a Trigger Script to execute. FIG. 3, as discussed below, outlines what happens within the Action Manager from the point it receives the trigger identifier through the trigger notification interface from the Trigger Provider. This interface 302 has two implementations in this embodiment of USIS: a) a C++ method exposed by the Action Manager within a software module (that is, within the application containing the Action

Manager); and b) a proprietary OLE Automation event exposed by compliant Trigger Providers to the Action Manager.

Trigger Provider Proxies, in this embodiment, are implemented as OLE Automation controllers and OLE Containers. The application containing the Action Manager is an example of the OLE Container type of Trigger Provider Proxy. As a proxy, it supports both Action Manager interfaces 301, 302 and interfaces 310 required by specific kinds of non-compliant Trigger Providers--in this case, OLE Automation objects that do not support the proprietary OLE events implement in this embodiment of the Action Manager. The Proxy enables non-compliant Trigger Providers to participate in the system by assigning OLE events in the non-compliant component to unique DWORD values, which the Proxy then uses in protocols compliant to the Action Manager. For example, the OLE Container version of the Trigger Provider Proxy would query (through conventional OLE methods) the OLE automation events provided by a non-compliant Trigger Provider. This Proxy would then assign a unique identifier to each event, publish the events to the Action Manager through a compliant interface 301, and wait for a notification of the occurrence of one of the events through conventional OLE protocol. Upon receiving a notification of an event from the non-compliant Trigger Provider, the Proxy then notifies the Action Manager through a compliant interface 302. From that point, the flow of action is identical to that which occurs with a compliant Trigger Provider.

Action Providers: In the preferred embodiment of the invention, compliant Action Providers take the form of a) OLE automation controls and b) the application containing the Action Manager object. These controls maintain a collection of software-meaningful action identifiers, each of which is associated with a specific command or method. Action Providers make themselves visible to the Action Manager via the action publication interface 303, through which they publish a string value and a DWORD value identifying an action to the Action Manager object 200, which stores these values in the Action Pool 202 as described above. This interface 303 has two implementations in this embodiment of USIS: a) a set of C++ methods exposed by the Action Manager within a software module (that is, within the application containing the Action

Manager); and b) a proprietary, but publicly accessible, section in the Windows operating system registry.

The Action Manager executes actions stored in a Trigger Script via the activation interface 304, through which the Action Manager invokes the methods stored in Trigger Script in an Action Provider. FIG. 3 as discussed below outlines how the Action Manager determines what action to invoke in what Action Provider in the process of executing a particular script. This interface 304 has two implementations in this embodiment of USIS: a) a proprietary command message for invoking actions from within a software component (in this case a type of window message as might be used in typical Windows based applications); and b) a proprietary OLE Automation method exposed by compliant Action Providers.

Action Provider Proxies, in this embodiment, are implemented as OLE Automation controllers and OLE Containers. The application containing the Action Manager is an example of the OLE Container type of Action Provider Proxy. As a proxy, it supports both Action Manager interfaces 303, 304 and interfaces 310 required by specific kinds of non-compliant Action Providers--in this case, OLE Automation objects that do not support the proprietary OLE events implemented in this embodiment of the Action Manager. The Proxy enables non-compliant Trigger Providers to participate in the system by assigning OLE events in the non-compliant component to unique action identifiers, which the Proxy then uses in protocols compliant to the Action Manager. For example, the OLE Container version of the Action Provider Proxy would query (through conventional OLE methods) the OLE automation events provided by a non-compliant Action Provider (for example, Microsoft's Web browser control). This Proxy would then assign a unique identifier to each OLE method, publish these methods to the Action Manager through a compliant interface 303, and invoke the method when necessary (that is, when the Action Manager executes an action the proxy through a compliant interface) through conventional OLE protocols. Until an action in the Proxy is invoked through a compliant interface, the flow of execution is identical to that which occurs with a compliant Action Provider.

Referring to FIG. 2, which is a flow diagram illustrating the method 204 of configuring the action and trigger objects using USIS shown in FIG. 1 in the user-

definition layer 205 section. A user begins the configuration at step 40 by assigning trigger objects to the assigned action objects from the Trigger Pool and Action Pool. A user selects at least one trigger at step 42 and an action object from the Action Pool at step 44. At step 46, a determination is made whether there are any arguments. If there are arguments, then specify that argument for that action object until there are no more arguments to specify at step 50. At step 52, specify the Action Provider. At step 54, determine whether there are anymore action objects to assign and if so, go to step 44. If there are no more action objects to assign, go to step 56 and save the script for a specific trigger instance if selected at step 58 in the Collection of Trigger Scripts 203. At step 60, determine whether to save a script for a Trigger Provider Class at step 62 in the Collection of Trigger Scripts 203, if not go to step 64 and determine if there are more Trigger Objects to configure in USIS. If no more Trigger objects are to be assigned, then end the process at step 66.

Referring to FIG. 3, which is a flow diagram illustrating the method 206 of executing the action and trigger objects using USIS shown in FIG. 1 in the software interaction layer 207 section. A user begins the execution of a trigger object at step 70 when the Action Manger 200 receives a user specified input event associated with the assigned trigger object previously defined in the method 204 in FIG. 2. At step 72, get Trigger Provider Class, Instance, Trigger ID (the parameters). At step 73, find the Script for the Instance, Class or Default from the Collection of Trigger Script 203. At step 74, a determination is made whether a Trigger Script is found for the Class Instance, Class or Default and if not, the execution ends. If there is a Trigger Script, go to step 76 and obtain the Action object from the Script (Action Provider ID and Action ID) and then proceed to step 77 and determine whether are arguments. If there are arguments, then get and set the argument for the action in step 78 until there are no more arguments as determined by step 79. Then at step 80, find an Instance of the Action Provider (New, first available, or a specific) and then proceed to step 82 and execute the Action Object. At step 84, determine whether there are anymore actions in the Script to execute. If there are more Actions to execute, go to step 76 again until there are no more Scripts. At step 86, the execution process ends.

FIG. 4 illustrates an exemplary and preferred use of the invention, which is part of a system and user-environment that provides financial data services to subscribers through desktop terminals from commercial financial information databases. The terminals can also display other information received from local connections independent of these commercial databases. Trading information from various exchanges, news services, and private databases typically flows into a national computer center 10 over conventional means (not shown). In the national computer center the information is processed to form a file of transaction and quotation data.. Derived files of statistics on individual securities and markets are also maintained. Additional files of data useful to a subscriber are maintained, including dividend, earnings, and forecasting information for a variety of financial instruments, stocks and bonds.

The national computer center 10 is connected through an information transmission structure 12 including transmission lines, regional data centers, and concentrator sites (all not specifically shown). Other services, independent of the financial data, are provided from third party services 13 by conventional means of data transmission. At a typical subscriber's site, a subscriber server 14, which can be multiple servers collectively shown as the server 14, is connected to the information transfer structure 12 and third party services 13 and through a local area network 15 (or the Internet) to a plurality of branch terminals 16. Three are shown but there could be any practical number. Preferably, the server 14 is a NT or UNIX-based machine executing either NT or UNIX operating systems with an appropriate application program interface (API) and the terminals have a Microsoft Windows-based operating system in personal computers (PC) which run a commercially-available user interface such as the Microsoft Windows NT system. The architecture of a branch terminal such as a terminal 16. NT and UNIX are trademarks of their respective corporate entities.

Referring to FIG. 5, the terminal 16 typically includes a commercially-available PC 17, which is capable of supporting concurrent execution of a plurality of application programs 18, 19, and 20. The application programs 18-20 are interfaced through an OLE container applications 22 that is joined conventionally to a graphical user interface 24. The graphical user interface is a software system which allows a user to directly manipulate the application programs 18, 19, and 20 by means of conventional I/O

devices such as a CRT display 25, a keyboard 26, or a mouse 27, or other user specified device, or all of them if desired. A file manager 30 is alsoprovided which opens, maintains, and closes files on behalf of the navigator function 22 and the graphical user interface 24. The file manager 30 operates in conjunction with one or more peripheral storage devices such as a direct access storage device (hard disk drive) 31.

Referring to FIG. 5, the GUI 24 of the present invention is shown in connection with graphical display and control of executing multiple active applications, and in particular, financial service applications. In the preferred embodiment of the navigator function 22, the Reuters Plus (a trademark of Reuters Limited) financial service navigator is used on a platform operating system with GUI capability, preferably Microsoft Windows 95/98 or NT GUI. The navigator function 22 and interface 24 manage and control the contents of the "container" having multiple active application modules. Although the preferred embodiment of the invention has a user interface 24 based on Microsoft Windows operating system, the container can be implemented in any number of different GUI operating systems.

The navigator function application 22 is a software program written in the well-known C++ language and is capable of being compiled and executed on a PC processor such as the processor 17. The navigator application function 22 can include macro-instructions called a keystroke or mouse "hook" which interfaces the navigator function 22 with the keyboard 26 or preferably uses a mouse 27 or other user defined input (for example, voice recognition). This function can be understood with reference to the Programmer's Reference: Functions, published by Microsoft Corporation (1987-1998) for the Microsoft Windows operating system. As an example of a user interface input control, either a keystroke or mouse control hook function intercepts all input keys or mouse actions and passes them to a keyboard hook or mouse processing routine in the navigator function 22 for processing before they are passed to theapplications. An exemplary form for a navigator uses the keystroke hook function that enables the navigator function 22 to detect user specified actions in response to which the navigator function invokes certain actions. The keystroke hook function is discussed in detail in commonly assigned U.S. patent 5,721,850 entitled, "Method and means for navigating user interfaces which support a plurality of executing

applications." The preferred navigator function 22 is the container that encompasses multiple applications using Microsoft ActiveX/OCX "object" technology for implementing these application objects. It is understood, however, that application objects can be from any application that meets the API specifications of the GUI.

The terminals 16 can include a market quote application which is a container for displaying multiple service applications (objects). In the preferred embodiment of the navigator function 22, the Reuters Plus financial service navigator 22 is used on a platform operating system with GUI capability, preferably a Microsoft Windows 95/98 or NT GUI. These multiple financial service application modules are ActiveX object designs using the API of Microsoft Windows operating system, which are linked to databases over the LAN 15. These objects can include multiple financial information services from sources such as Dow Jones News, Reuters, EDGAR SEC Filings and the like and Internet Web sites such as Yahoo (EDGAR and Yahoo are trademarks of the Securities Exchange Commission and Yahoo, Inc.). These applications are listed in the application table of the navigator function 22. Other services available from third parties may be provided to the terminals either through the system structure 10, 12 of FIG. 4 or by the third party services 13. These third party services are supported by third party applications, such as the application 20, that co-executes with the navigator function 22 in a terminal. Third party programs may include, for example, the Microsoft WORD and EXCEL programs (WORD andEXCEL are trademarks of Microsoft Corporation). The navigator function 22 also maintains a system configuration file. The system file is maintained for the purpose of establishing a set of window characteristics definitive of a set of system windows and is used as a point of reference in navigating through window configurations. In addition, each application (including the navigator function) maintains its own window configuration buffer which is used, when the application executes, to keep the current parameter values definitive of the current configuration of the application's window. Such buffers include window configuration buffers and are maintained in real memory. Various methods and apparatus currently exist for allowin a GUI to support many different applications. Furthermore, portions of data, or objects, can be shared between the different applications. For example, the third party services as illustrated in FIG. 4 can be

applications that interface with other applications such as a Microsoft EXCEL spreadsheet application as well as word processing application such as Microsoft WORD word-processing program. Using Microsoft Windows, a selection of spreadsheet cells of a Microsoft EXCEL spreadsheet can be placed inside a WORD document. In this example, the EXCEL spreadsheet is an "object server" document, the selection of spreadsheet cells is an "object," and the WORD document is an "object container" document. The WORD container document is a conventional document, except that it provides a "client" window for the object. As a result, the object is either linked or embedded into the container. When an object is linked from the server document to the container document, a user may edit the object by making changes to the server document. When the object is embedded from the server document to the container document, the user may edit the object inside the container document by accessing features of the server document's application.

Referring to FIG. 6, a financial service navigator application window 100 is shown, and in particular, the Reuters Plus (a trademark of Reuters Limited.) application made by Reuters, New York, NY. This application provides current market quote data to financial service professionals. This window typically could appear on a terminal display 25. The navigator window 100 includes a menu bar 102, an application object execution button bar 104, and a display areas of multiple financial application objects 106, 108, and 110 on a display screen. Exemplary application objects 106, 108 and 110 contain research, market data news fundamentals and analytics in one window. An Internet Web browser 116 can be incorporated in the container as one of the application objects and shown in part of the display as seen navigated to the Yahoo web site. Each of the buttons in the bar 104 represents particular object window viewing for a selected financial service application. Adjacent the top of the application object display are button tabs 112 for control of the container, each of which identifies a functionality such as printing, searching services and the like within an one of application object programs. In exemplary form, the invention USIS method can be executed by activation of the view properties menu item 120 by "clicking" thereon using a mouse or other user specified input device.

The service application objects 106, 108 and 110 are controllable objects using the Microsoft NT OCX container-type programming. These applications include a Web browser capability with an array of third-party information sources, combined with real-time news and market quote data package that reflect several segments of the US equity market. This application allows a user to integrate their own proprietary functions with Reuters Plus application. By combining market data and third party information providers with a customer's intranet and the Internet, a user can access all this information simultaneous on a single screen. A number of different third-party providers include: fundamental data on US stocks from Market Guide, mutual fund data from CDA/Wiesenberger, CDA/Spectrum institutional holdings data, earnings estimate data from IBES and Disclosure's EDGAR service. Other features of the Reuters Plus application can include 32-bit-type container applications that support client and third party applications such as Microsoft EXCEL spread sheet application and WORD word processor application, Reuters News and Dow Jones News Service commingled and searchable with Reuters news Internet browser included in the Reuters Plus application to allow seamless integration of company intranet and Internet information with market quote data.

The Reuters Plus application containing financial market quote data also provides personalized action management of events by allowing a user (for example, a broker) to create unique association of triggering events with a series of resulting actions. These customized action management triggering events effectuate any combination of service application action objects (for example, these include displaying of Chart History of a security, and then sending the chart to a printer or fax; going to a user designated display page, and requesting on that page a quote, headlines, and information from EDGAR filings). These actions are assigned to a triggering event (for example, a double clicking on a field, a hotkey or combination of hotkeys, a generated active icon button, a balloon menu associated with a particular Active-X object in an application object). To initiate execution of the USIS method, a user clicks on the Setup control button 120 in the menu bar of the Reuters Plus application 100.

In FIG. 7, an Action Manager window 150 of Reuters Plus application is shown where user selects a trigger, which in this case is a double clicking on a field "Block

Count" in one of the trigger providing modules. This window also allows a user to select a trigger and configure, or assign a series of actions to it.

In FIG. 8, an Action Manager window 160 of Reuters Plus application is shown where user assigns or removes an assignment of a series of actions to the previously selected trigger in FIG. 7, the trigger, which is double clicking on "Block Count" field.

In FIG. 9, an Action Manager window 170 of Reuters Plus application is shown where user assigns a series of available action to the previously selected hotkey "F9" (not shown). This window allows a user to edit previously configured triggers.

Many users of Reuters Plus utilize the USIS in order to facilitate their trading activities. As an example, a broker selects a field inside a quote service module, and configures a double-click trigger for that field to produce a display of historical chart, headlines, fundamental data, and send this display to a printer. While consulting with a customer from a Reuters Plus workstation, the broker can double-click on the specified trigger field, and the desired actions will follow. As another example, a broker selects a conditional trigger of an alert set to indicate that a security of interest reached a specified price. The broker configures this trigger to produce a chart display for this security, print it, and also send a message via e-mail to his or her associates and pager device. If a security of interest reaches the price specified by the trigger configuration, even in the case that the broker is away from a Reuters Plus workstation, an alert will trigger the desired series of actions: a printer will print a chart for that security, the broker's associates will get an e-mail tailored to the triggering event, and the broker will receive a pager message.

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. Adaptation and optimization, as well as others within the abilities of those of skills in the art, may be performed on the system and methods disclosed herein without departing from the scope and spirit of

the present invention. Consequently, the scope of the invention is not limited to the specific examples given herein but is set forth in the appended claims.

CLAIMS

What is claimed is:

1. In a computer system that having a digital data processor, a display coupled to the processor, and a user interface coupled to the processor and the display; a method for selecting and executing user-defined programming interactions within the system, the method comprising the steps of:

a) initiating a user defined action manager process for accessing both: i) an action pool comprising multiple action objects, and ii) a trigger pool comprising multiple trigger objects, wherein subsequent execution of a selected trigger object causes execution of associated selected action objects, and each selected action object comprises executable object oriented interacting programming within the computer system;

b) selecting and inputting to the user interface at least one of the trigger objects to define a triggering event;

c) associating at least one action object with the triggering event;

d) monitoring and detecting for the triggering event by the processor; and

e) executing the at least one action object defined in step c).

2. The method of claim 1, wherein step a) includes providing the trigger objects that comprise inputting user-defined interaction to the user interface by an input device selected from the group consisting of a mouse, keyboard, touchpad, and voice recognition device.

3. The method of claim 1, wherein step a) includes providing the trigger objects that comprise inputting any combination of previously user-defined parameter monitoring input from executing applications that include information from network data streams, computer system operating messages, application interaction protocols and scheduling programs having alarm broadcasts.

4. The method of claim 1, wherein step a), the trigger objects comprise inputting any combination of previously user-defined parameter monitoring input from status changes in hardware configuration of the computer system that include a data link connection, a computer clock generated event, and a status of operation of applications within the computer system.

5. The method of claim 1, wherein step a) includes providing the action objects that comprise inputting any combination of programmable functions that include an operating system of the computer system, and application programming interface protocols that include creating objects through operating control extensions (Active X/OCX objects).

6. The method of claim 1, wherein step a) includes providing the action objects that comprise inputting any combination of programmable functions for executing specific modules capable of calling exported functions using object linking and embedding (OLE) mechanisms, dynamic data exchange (DDE), dynamic link libraries (DLLs), and functions that establish OLE properties.

7. The method of claim 1, wherein step a) includes providing and controlling access to the action pool by an action provider.

8. The method of claim 7 includes providing the action provider with compliant capability.

9. The method of claim 7 includes providing the action provider with non-compliant capability through a proxy.

10. The method of claim 1, wherein step a) includes providing and controlling access to the trigger pool by an trigger provider.

11. The method of claim 10 includes providing the trigger provider with compliant capability.

12. The method of claim 10 includes providing the trigger provider with non-compliant capability through a proxy.

13. The method of claim 1, wherein step b) and step c) comprise steps for configuring trigger scripts that include:

aa) determining whether there are arguments associated the action object by using the action manager process;

bb) specifying an action provider;

cc) repeating step aa) and step bb) if there if there is another action object associated with the at least one trigger object; and

dd) saving a trigger script associated with the trigger object.

14. The method of claim 1, wherein step d) and and step e) comprise steps for executing triggering scripts that include:

ee) receiving the triggering event by the processor;

ff) obtaining a trigger provider associated with the triggering event;

gg) obtaining a trigger script associated with the trigger provider; and

hh) obtaining and executing the at least one action object associated with the trigger script.

15. The method of claim 1, wherein the providing of a computer system with the user interface, the system including a container application that comprises a financial information service application with multiple application objects comprise real-time market quote data from multiple sources, the method includes providing application program interfacing of the action manager process with the container application.

16. A computer system having programming interaction management, the system comprising:

a display;

a user interface; and

a digital data processor, coupled to the display and user interface, and

the processor includes:

means for initiating a user defined action manager means for accessing both: i) an action pool comprising multiple action objects, and ii) a trigger pool comprising multiple trigger objects, wherein subsequent execution of a selected trigger object, which causes execution of associated selected action objects, and each selected action object comprises executable object oriented interacting programming within the computer system;

means for inputting to the user interface at least one of the trigger objects thereby defining a triggering event;

means for associating at least one action object with the triggering event;

means for monitoring and detecting the triggering event by the processor; and

means for executing the at least one action object defined by the means for associating the at least one action object with the triggering event.

17. The method of claim 16, wherein the means for inputting to the user interface at least one of the trigger objects to define a triggering even and the means for associating at least one action object with the triggering event comprise:

means for determining whether there are arguments associated the action object by using the action manager process;

means for specifying an action provider; and

means for saving a trigger script associated with the trigger object, thereby configuring the trigger script.

18. The method of claim 16, wherein the means for monitoring and detecting the triggering event by the processor and the means for executing the at least one action

object defined by the means for associating at least one action object with the triggering event comprise:

means for receiving the triggering event by the processor;

means for obtaining a trigger provider associated with the triggering event;

means for obtaining a trigger script associated with the trigger provider; and

means for obtaining and executing the at least one action object associated with the trigger script, thereby enabling execution of the trigger script.

19. The system of claim 16, wherein the computer system comprises a local area network (LAN) coupled to a terminal that encompasses the digital data processor, the LAN attaches to a service provider.

20. The system of claim 19, wherein the service provider includes multiple financial data bases that provide real-time financial market quote data.

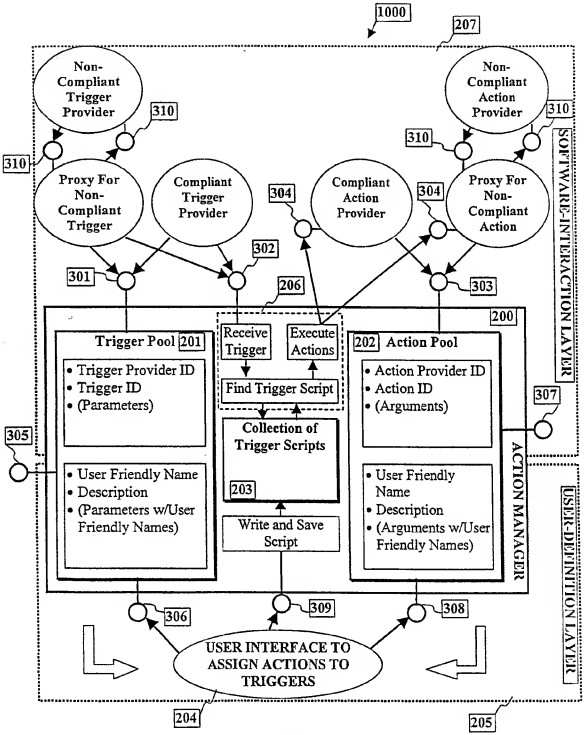


FIG. 1

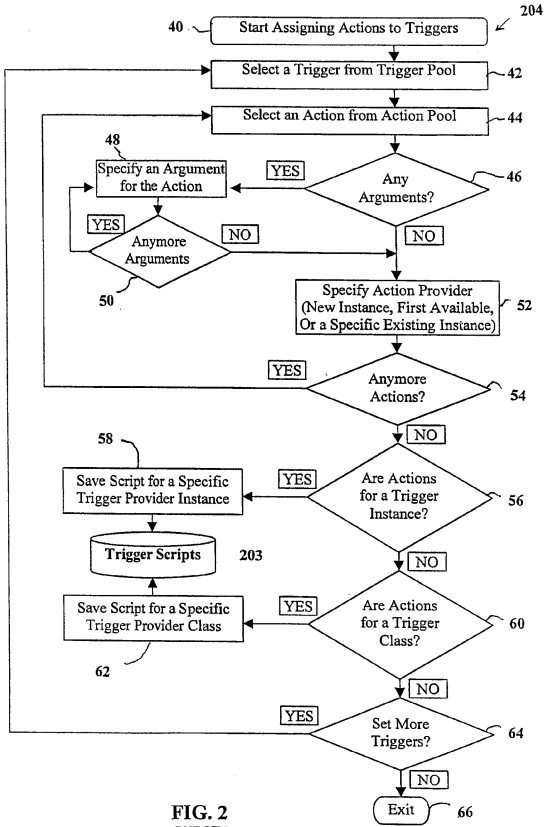


FIG. 2

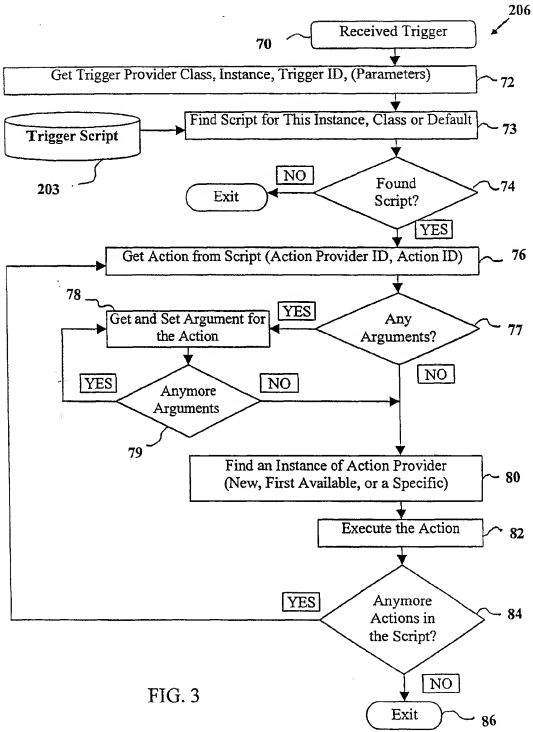


FIG. 3

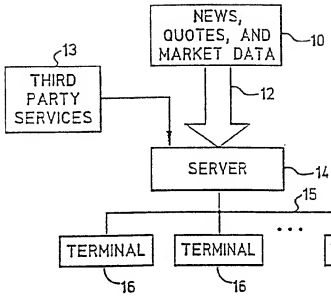


FIG. 4

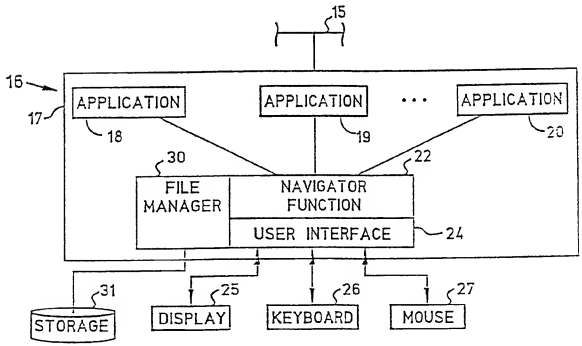


FIG. 5

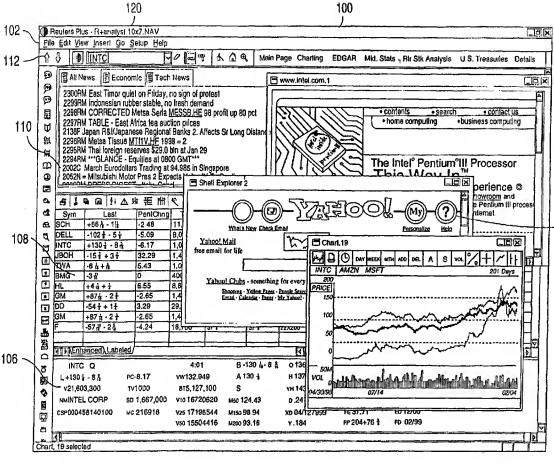


FIG. 6

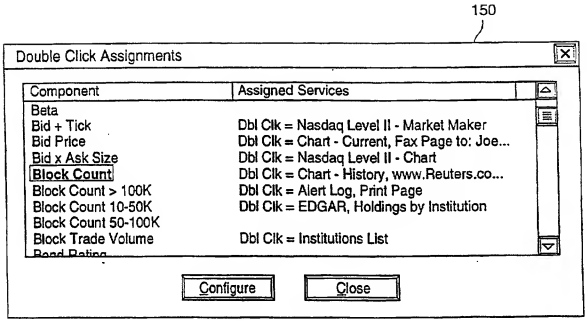


FIG. 7

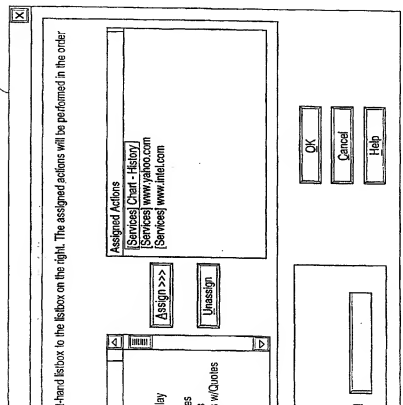


FIG. 8

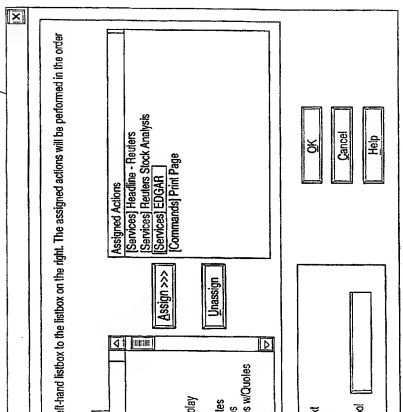


FIG. 9

INTERNATIONAL SEARCH REPORT

Interna...nal Application No
PCT/US 99/02801

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/44 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 451 963 A (IBM) 16 October 1991	1-4,7, 10,13, 14,16-18
A	see column 1, line 33 - column 2, line 37 see column 4, line 14 - column 5, line 22 see column 6, line 8 - line 26; figure 8 see column 6, line 50 - column 7, line 24	8,9,11, 12,15, 19,20
X	WO 95 31768 A (APPLE COMPUTER ;KATZ GLENN (US); GOUGH MICHAEL LANE (US); JACOBS J) 23 November 1995	1-4,13, 14,16-18
A	see page 3, line 1 - page 4, line 3 see page 8, line 23 - page 12, line 2 see page 23, line 14 - line 25 see claims 1,3-5	7-12,15, 19,20

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
29 June 1999	06/07/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Bijn, K

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Intern. nal Application No
PCT/US 99/02801

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X A	WO 96 31823 A (SOFMAP FUTURE DESIGN CO LTD) 10 October 1996 see page 15, line 4 - line 18 see page 22, line 23 - page 25, line 26 see page 45, line 22 - page 47, line 13; figure 17	1-6,13, 14,16-18 7-12,15, 19,20

3

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

EP 308971 (1)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Intern. al Application No
PCT/US 99/02801

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
EP 0451963	A	16-10-1991	JP	2554381 B	13-11-1996
			JP	3282862 A	13-12-1991
			CA	2038265 A	01-10-1991
			US	5522024 A	28-05-1996
			US	5600780 A	04-02-1997
WO 9531768	A	23-11-1995	AU	2551195 A	05-12-1995
			EP	0760123 A	05-03-1997
			JP	10500510 T	13-01-1998
WO 9631823	A	10-10-1996	AU	5121296 A	23-10-1996
			CN	1181141 A	06-05-1998
			EP	0819274 A	21-01-1998
			JP	9319565 A	12-12-1997
			US	5799181 A	25-08-1998